Master's Thesis

Data Locality Optimization on Resource - Shared Clusters

Pham Kieu Thao Nguyen

Department of Computer Science and Engineering

Graduate School of UNIST

2018

Data Locality Optimization on Resource - Shared Clusters

Pham Kieu Thao Nguyen

Department of Computer Science and Engineering

Graduate School of UNIST

Data Locality Optimization on Resource - Shared Clusters

A thesis
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Pham Kieu Thao Nguyen

12/12/2017 of submission

Approved by

Advisor
Young ri Choi

Data Locality Optimization on Resource - Shared Clusters

Pham Kieu Thao Nguyen

This certifies that the thesis of Pham Kieu Thao Nguyen is approved.

Month/Day/Year of submission

signature

Advisor: Young-ri Choi

signature

Beomseok Nam: Thesis Committee Member #1

signature

Woongki Baek: Thesis Committee Member #2

signature

Abstract

Resource-shared clusters between users on various computing frameworks such as Dryad, Hadoop, and Spark are becoming a hot trend mainly due to the lower costs offered when input data are shared among separated clusters. However, the performance of such resource-shared clusters relies on data locality, which is directly related to the overhead of the communication network.

Due to this issue, we decided to study the scheduling methods to achieve an optimal solution for data locality, enabling a better performance of applications on the resource-shared clusters while still ensuring the fairness. Our first approach was based on the Linear Programming algorithm, while the second one was adopted from the graph algorithm - Min Cost Max Flows.

To demonstrate the applicability of our solutions, an experimental study was carried out, using various workloads consisting of multiple Hadoop and Spark applications. The results showed that our approaches improved the performance of these workloads, namely 1.67 times faster than conventional static partition and 1.49 times faster than fair sharing methods.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Instead of using private clusters for different frameworks such as Dryad[1], Hadoop[2] and Spark[3], public resource-shared clusters like Mesos[4] have become popular due to many advantages. First, the utilization between applications is improved since sharing enables statistical multiplexing. Second, a resource-shared clusters also allows applications to share access to large datasets that may be too costly to replicate across clusters.

While the scheduler of public clusters determines how many resources are allocated to each application, the application is responsible for which resources to be assigned to which tasks and when. Different resource assignment options can lead to significantly different outcomes in terms of scalability, performance, and so on. As the size of a cluster grows, the network can bottleneck if the assignment resources are not placed closely to the application's input data. Therefore, the schedule is an important aspect in the resource-shared cluster.

Many studies relating to improvement of the scheduler's performance have been conducted. Some examine all of the cluster states to determine the most suitable resource for in-coming applications, such as Quasar [5] and Quincy[6]. Quasar uses classification to determine the resource state that meets the application's requirement with the minimal contention by a greedy scheduler without considering data locality. For every scheduling event, classification and the greedy search was repeated, which could lead to overhead on the really large cluster. In addition, Quincy formulates scheduling as a cost optimization problem that accounts for preferences with respect to locality, fairness, and starvation-freedom. However, Quincy just improves schedul-

ing throughput but not latency, and utilizes the whole node resource assignment for just one application.

In this study, we suggested that the optimization of the resource assignments could be archived by maximizing data locality. Two techniques based on Linear Programming(LP) [7] and Min Cost Max Flows(MCMF) [8, 9] were employed.

The rest of this paper is organized as follows: in Chapter 2, the feasibility of data locality on job performance is analyzed. Background and direction for archiving data locality public clusters is presented in Chapter 3. Afterwards, our methods and performance evaluation are presented in Chapter 4 and 5, respectively. In Chapter 6, works most relevant to ours are compared and a conclusion is presented.

Chapter 2

Data Locality

Data locality stands for the technique that implements the application in the place where its data resides. One data set is divided into some number of small blocks that will be stored across the data nodes in HDFS[10]. When one job or application is executed on those clusters, the job will also be divided into smaller tasks that will be implemented concurrently on different nodes. If the data is not available on the same node where the task is being performed, the data needs to be transferred over the network and would require more cost. Therefore, data locality is an essential factor for job performance.

In this chapter, we discuss how data locality affects application's performance by running some of the same single applications on two different clusters and comparing the results. The first test cluster consisted of one name node and four data nodes, and every single application was run directly on these nodes. The second test cluster had one name node and four data nodes, and an additional of four more nodes in which applications were run on a remote control to the HDFS. Each node was configured by GenuineIntel processors that have two sockets with 16 cores and an available local memory of 30 GB. Each application was run with 8 GB and 32 GB input size.

The block size of HDFS is 128 MB, and the replication factor is three. For Hadoop, the amounts of memory configured for a map task, a reduce task, and a node manager are 1GB, 2GB, and 16 GB, respectively. The default values are used in the case of Hadoop configuration parameters.

Figures 2.1, 2.2 and 2.3 illustrate the results of direct running and remote-control running for

Grep, Sort and Word Count. The experiment results show that the running time of the remote-control running is always lower than direct running. Furthermore, the difference between two running-time lines in Figures 2.1, 2.2 and 2.3, become more obvious when input size increases. In fact, the difference in running time between the two methods for 32 GB input size is nearly two times higher than that of 8 GB input. Due to this significant difference between the direct and remote control methods, we decided to optimize data locality in the allocation application on the resource-shared clusters as an attempt to improve the cluster's performance.

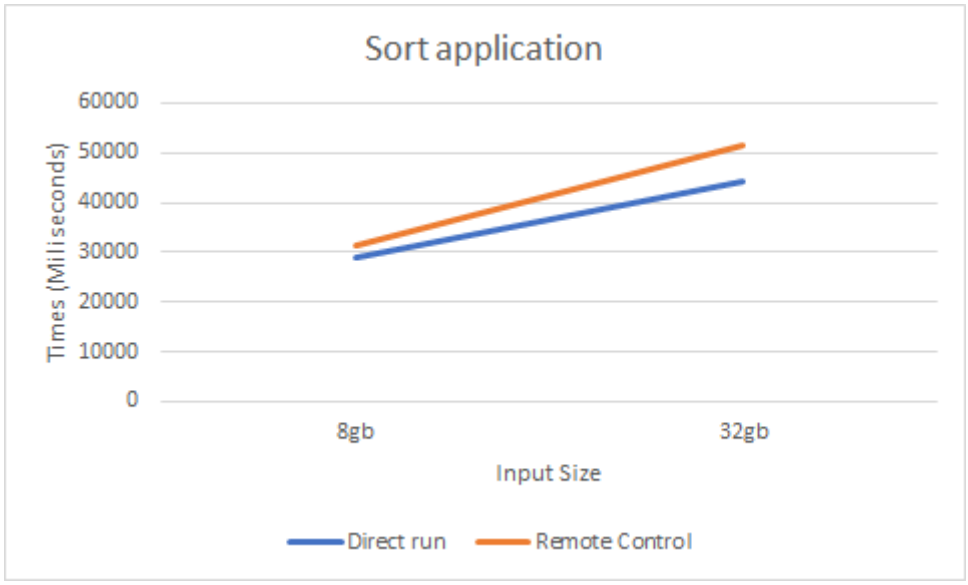

Figure 2.1: Data locality of Sort

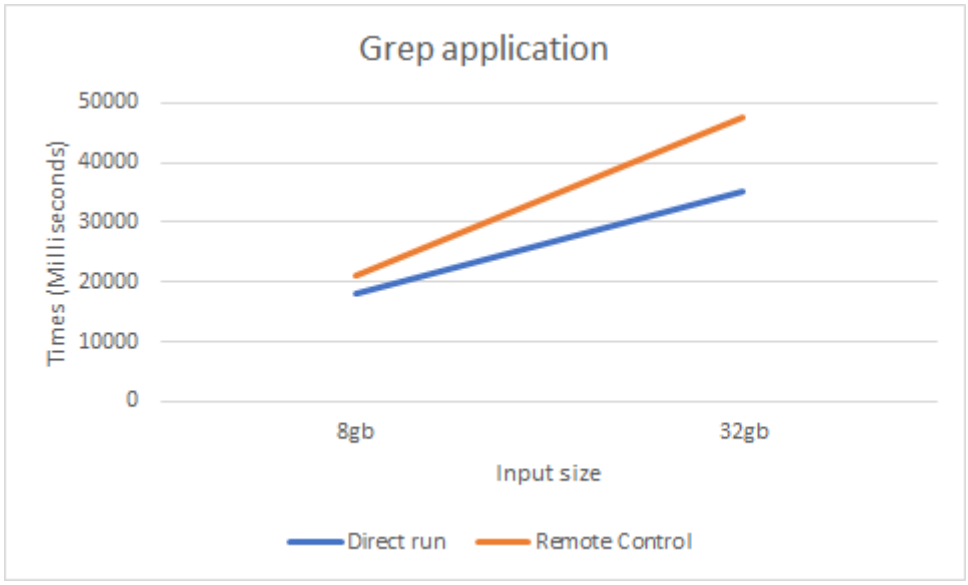

Figure 2.2: Data locality of Grep

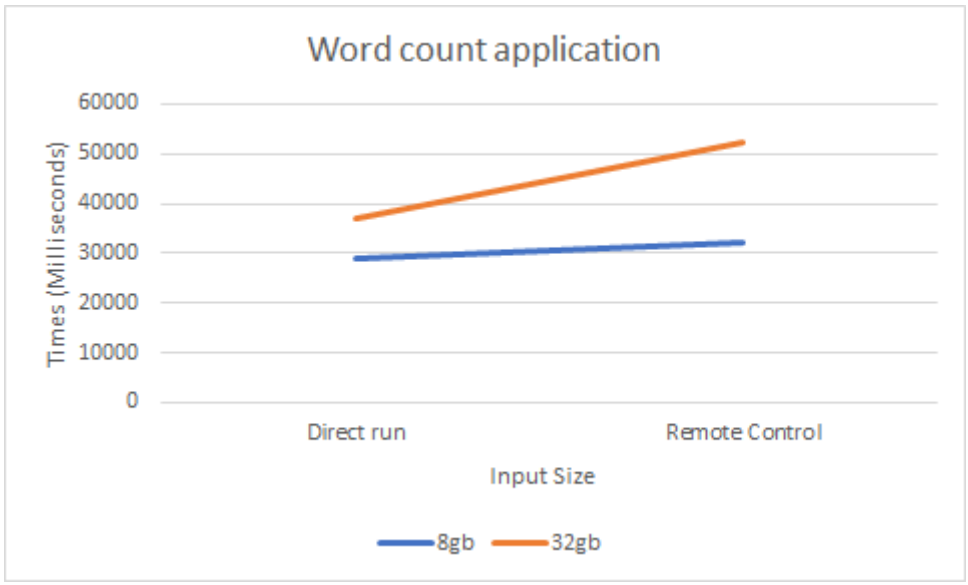

Figure 2.3: Data locality of Wordcount

Chapter 3

Background Theory

3.1 Linear Programming (LP)

LP[8], also known as linear optimization, was developed during World War II to solve the complex planning problem relating to the optimization of a linear objective function. George Danntzig [11] was the first researcher to develop the Simplex method in order to optimize the programming problem with a linear structure. The standard form of a linear program can be expressed as:

$$\begin{aligned} \text{Minimize} \quad & c_1x_1 + c_2x_2 + \ldots + c_nx_n = z \\ \text{Subject to} \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ & \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \\ & x_1, x_2, \ldots, x_n \geq 0. \end{aligned}$$

In LP, $c_1, c_2, \ldots, c_n, a_{11}, a_{12}, \ldots a_{mn}$ are given and z, the expression being optimized, is called the objective function. If maximization of the objective function is desired instead of minimization, the sign of $c_1, c_2, \ldots, c_n$ will be reversed. Variables $x_1, x_2, x_3, \ldots, x_n$ are known as decision variables, and their values are subject to $m + 1$ constraints (m line constraints and their boundary). A feasible point is where a set of $x_1, x_2, x_3, \ldots x_n$ satisfies all the constraints and the feasible region is the set of all such points. As a result, the solution of the LP must be a

point in the feasible region.

A linear program can take many different forms. We have minimized or maximized problems depending on the objective function, as in the formula shown above. In linear algebraic form, the problem can be written as:

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{n} c_1 x_1 = z \\ \text{Subject to} \quad & \sum_{i=1}^{n} a_{ij} x_i = b_i,\ j = 1, \ldots, m \\ & x_1, x_2, \ldots, x_n \geq 0. \end{aligned}$$

In matrix form, a linear program becomes:

$$\begin{aligned} \text{Minimize} \quad & z = c^T x \\ \text{Subject to} \quad & Ax = b \\ & x \geq 0. \end{aligned}$$

where $c = (c_1, c_2, c_3, \ldots, c_n)^T, b = (b_1, b_2, \ldots, b_n)^T$, and $x = (x_1, x_2, x3, \ldots, x_n)$ are column vectors, and are named as coefficient vector, right-hand side vector and solution vector, respectively. c^T stands for the transposition of column vector c, and A is an $m \times n$ matrix. Any linear program can be expressed into an equivalent linear program in the standard form. As further qualification, the following applies:

- If we have an inequality constraint $a_{i1}x_1 + \ldots + a_{in}x_n \leq b_i,\ i \in \{1, m\}$, then we can transform it into the equality constraint by adding a non-negative slack variables, say t, such that: $a_{i1}x_1 + \ldots + a_{in}x_n + t = b_i$ and $t \geq 0,\ i \in \{1, m\}$
- Similarly, if we have an inequality constraint $a_{i1}x_1 + \ldots + a_{in}x_n \geq b_i,\ i \in \{1, m\}$ then we can transform it into an equality constraint by adding a non-negative surplus variables, say t, such that: $a_{i1}x_1 + \ldots + a_{in}x_n + t = b_i$ and $t \geq 0,\ i \in \{1, m\}$

There are two popular methods to solve LP problems with regards to simplex and interior methods[12]. The simplex passes from the vertex to the vertex on the boundary of the feasible

polyhedron, and then repeatedly increases or decreases the objective function until either an optimal solution is found or no solution exists. Linear programs in many variables are solved using the simplex method on modern computers due to their high efficiency in practice. Conversely, the interior point methods[13], developed by Leonid Khaciyan in 1979, is the ellipsoid method used to solve the problem in polynomial time.

3.2 Min Cost Max Flows (MCMF)

The minimum cost flow problem [8, 14] is a concept of minimizing the cost required to deliver the maximum amount of flow possible in the network. It is also known as one extension of the maximum flow problem but adding the constraint on cost per flow for each edge on the graph.

A flow network is a directed graph G = (V, E) defined by a set of V vertices or nodes and a set of E edges (arcs). Each edge $(i, j) \in E$ has capacity u_{ij} and an associated cost c_{ij} which denotes the maximum amount that can flow on that edge together with the cost per unit flow. Each vertex v furthermore has a demand $b_v \in R$. If $b_v \geq 0$ then v is known as a source, and if $b_v < 0$ then v is called a sink. Figure 3.1 shows the network parameters explicitly.

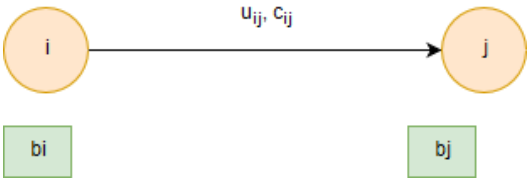

Figure 3.1: Min cost Max Flows graph

The minimum cost flow problem asks for flows on arc (i, j), x_{ij} that conserve the flow at each vertex, respect the upper and lower bounds (u_{ij} and 0, respectively), and minimize the overall cost. Formally, the optimal solution for a minimum-cost flow of G can be found using the following optimization problem:

$$\begin{aligned} \text{Minimize} \quad & \sum c_{ij}x_{ij}, \ \forall i, j \in E \\ \text{Subject to} \quad & b_i - b_j = \sum x_{ij} - \sum x_{ji} \ \forall i, j \in E \\ & 0 \leq x_{ij} \leq u_{ij}, \ \forall i, j \in E \end{aligned}$$

There are four popular algorithms to solve the MCMF problem including cycle canceling [15], successive shortest path[14], relaxation[16, 17] and cost scaling[18]. While cycle canceling, or ignoring the cost function first, computes the max flow solution and then uses a cycle guarantee as an overall solution to decrease cost, the successive shortest path algorithm maintains cost optimization at every step, repeatedly selects a source, and sends flows to sink along the shortest path. Similar to the successive shortest path, the relaxation solution augments the flow from source nodes along the shortest path but consider the dual problem. Cost scaling not only decreases the cost but also maintains feasibility using some slack variable. If N is the number of nodes, M is the number of arcs, C is the largest arc cost and U is the largest capacity, and the complexity of cycle canceling, successive shortest path, cost scaling and relaxation are $O(NM^2CU), O(N^2Ulog(N)), O(M^3CU^2)$ and $O(N^2Mlog(NC))$

Chapter 4

Methodology

In this section, we present how to formulate LP and MCMF to solve allocation on resource share clusters. Figure 4.1 depicts the process of using LP and MCMF for scheduling applications on public clusters. First, the block input information of these applications were extracted. Second, this information were converted to the form that LP and MCMF can use before applying these methods to find an allocated solution. Finally, the application will be executed on the above allocation on share resource clusters.

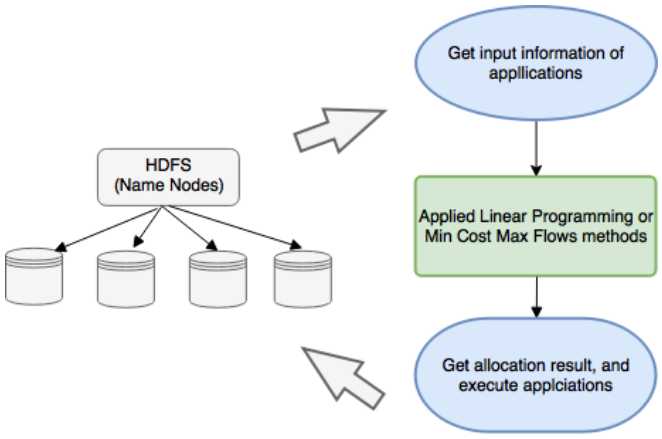

Figure 4.1: Process of using LP and MCMF

To formulate the LP and MCMF, information for the application input that we need to collect is the number of blocks each application has allocated in one node. Let us assume that J applications (jobs) will run at the same time on K nodes on resource shared clusters. Let us call B_j the number of input blocks of a job j

4.1 Linear Programming (LP)

Let us call d_{bk} the distance from a block b of a job j to certain node k.

- $d_{bk} = 0$ if block b stored on node k
- $d_{bk} = 1$ if block b stored on one node which is on the same rack with node k
- $d_{bk} = 2$ if block b stored on one node which is on different rack with node k

For a job j, let us call T_{jk} the value to estimate the sum of the distance of each block b of job j to node k

- $T_{jk} = \sum_{b=1}^{B_j} \sum_{k=1}^{K} d_{jk}$
- K is the total number of nodes
- B_j is the total number of blocks of job j
- d_{bk} is the distance a block b of job j to node k
- $D_{jk} = \frac{T_{jk}}{B_j}$ is the average distance from every block b of job j to node k

Input: $B_{J\times 1}$
Output: Matrix $D_{J\times K}$
initialization;
for $j = 1 \rightarrow J$ **do**
  for $k = 1 \rightarrow K$ **do**
    $D_{jk} = 0$
    for $b = 1 \rightarrow B_j$ **do**
      $D_{jk} = D_{jk} + d_{bk}$
    end
    $D_{jk} = \frac{D_{jk}}{B_j}$
  end
end

Algorithm 1: Computing D

Algorithm 1 illustrates the steps to compute D_{jk}. Let us call x_{jk} is the binary value that decide if the job j could be executed on node k or not. If x_{jk} equals 1, the application j will be implemented on node k, otherwise, it will not be allocated on node k. In addition, N will be denoted as the maximum number of nodes that application can execute, and A will be the

maximum number of applications running on one node .The linear programming problem will become:

$$\begin{aligned} \text{Minimize} \quad & \sum_{j=1}^{J}\sum_{k=1}^{K} D_{jk}x_{jk} = z \\ \text{Subject to} \quad & \sum_{k=1}^{K} x_{jk} = N.\ \ \forall j \in \{1, J\} \\ & \sum_{j=1}^{J} x_{jk} = A\ \ \forall k \in \{1, K\} \end{aligned}$$

Let us use one simple example. Considering that we have two jobs ($J = 2$) and 4 nodes ($K = 4$) on the same rack, and the location information is shown in Figure 4.2. In this example, we have:

$$T_{11} = 2 + 1 = 3, T_{13} = 2, T_{14} = 1 + 2 = 3$$

$$T_{21} = 1, T_{22} = 2$$

Therefore,

$$D_{11} = \frac{3}{4}, D_{12} = 1, D_{13} = \frac{1}{2}, D_{14} = \frac{3}{4}$$

$$D_{21} = \frac{1}{2}, D_{22} = \frac{1}{2}, D_{23} = 1, D_{24} = 1$$

Assume that N = 2 and A = 1. Our LP will become:

$$\begin{aligned} \text{Minimize} \quad & \sum_{j=1}^{2}\sum_{k=1}^{4} D_{jk}x_{jk} = z \\ \text{Subject to} \quad & \sum_{k=1}^{4} x_{jk} = 2.\ \ \forall j \in \{1, J\} \\ & \sum_{j=1}^{2} x_{ij} = 1\ \ \forall k \in \{1, K\} \\ & x_{jk} \in \{0, 1\}\ \ \forall j \in \{1, J\}, k \in \{1, K\} \end{aligned}$$

Or

$$
\begin{aligned}
\text{Minimize} \quad & D_{11}x_{11} + D_{12}x_{12} + D_{13}x_{13} + D_{14}x_{14} + D_{21}x_{21} + D_{22}x_{22} + D_{23}x_{23} + D_{24}x_{24} = z \\
\text{Subject to} \quad & x_{11} + x_{12} + x_{13} + x_{14} = 2 \\
& x_{21} + x_{22} + x_{23} + x_{24} = 2 \\
& x_{11} + x_{21} = 1 \\
& x_{12} + x_{22} = 1 \\
& x_{13} + x_{23} = 1 \\
& x_{14} + x_{24} = 1 \\
& x_{11}, x_{12}, x_{13}, x_{14}, x_{21}, x_{22}, x_{23}, x_{24} \in \{0, 1\}
\end{aligned}
$$

Or

$$
\begin{aligned}
\text{Minimize} \quad & \frac{3}{4}x_{11} + x_{12} + \frac{1}{2}x_{13} + \frac{3}{4}x_{14} + \frac{1}{2}x_{21} + \frac{1}{2}x_{22} + x_{23} + x_{24} = z \\
\text{Subject to} \quad & x_{11} + x_{12} + x_{13} + x_{14} = 2 \\
& x_{21} + x_{22} + x_{23} + x_{24} = 2 \\
& x_{11} + x_{21} = 1 \\
& x_{12} + x_{22} = 1 \\
& x_{13} + x_{23} = 1 \\
& x_{14} + x_{24} = 1 \\
& x_{11}, x_{12}, x_{13}, x_{14}, x_{21}, x_{22}, x_{23}, x_{24} \in \{0, 1\}
\end{aligned}
$$

The optimized solution of the above problem is $z = \frac{9}{4}$ where $x_{11} = 0, x_{12} = 0, x_{13} = 1, x_{14} = 1, x_{21} = 1, x_{22} = 1, x_{23} = 0, x_{24} = 0$. In our linear programming, we have a total of $N+J+N\times J$ constraints. In our experiment, we used CPLEX[19] to solve the problem.

4.2 Min Cost Max Flows (MCMF)

We define our problem based on the graph in Figure 4.3 with $b_i = N$ if vertex i represents the job and $b_i = -A$ if vertex i stands for the node. Figure 4.4 is the standard form view of MCMF

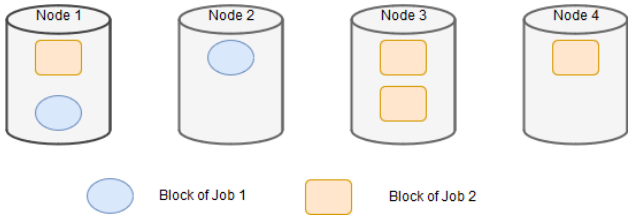

Figure 4.2: Block distributed of two jobs

in our problem . In this figure, sink s and source t are added. The capacity and cost from sink s to job i will be b_i or N and 0, respectively. The value of the capacity and cost from node i to t will equal $-b_i$ or A and 0, respectively.

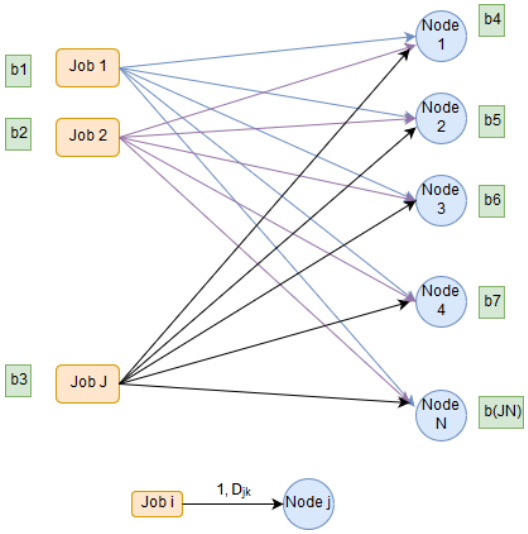

Figure 4.3: MCMF allocation graph

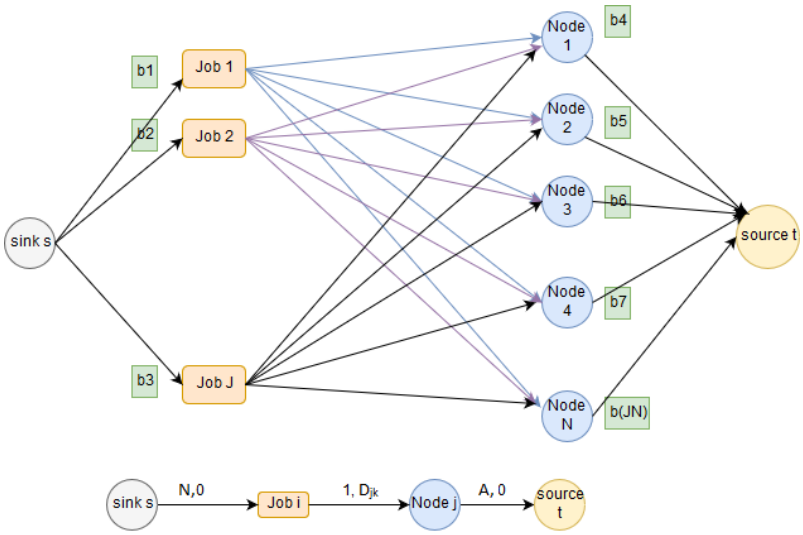

Figure 4.4: Standard form of MCMF allocation graph

Chapter 5

Evaluation

5.1 Experiment Setup

Our methods were evaluated on a small cluster composed of one master and 39 slave nodes. Each node were configured by Intel Xeon(R) E5530 2.40 GHz processors that had two sockets with 8 cores multi threading and an available local memory of 20 GB. Six workloads shown in Table 5.1 were used for validation. These workloads can be divided into three types, those with only Hadoop instances, only Spark instances and a combination between Hadoop and Spark instances.

Because LP and MCMF attempt to find the global solution, their allocation solution for the given input information must be similar in both cases. To study how allocation can affect the performance of applications on shared resource clusters, we measured the runtime of each application with the allocation of LP and MCMFs solutions and with the allocation of static and fair sharing methods. Since the runtime of each application is different, we had to maintain the environment by running the finished applications repeatedly until the last application in the batch finished. The input sizes of each application are shown in Tables 5.2 and 5.3

Overall, HDFS is shared between each user. The block size of HDFS is 128 MB and the replication factor is just one. For Hadoop, the amounts of memory configured for a map task, a reduce task and a node manager are 1 GB, 1 GB, and X GB, respectively, with X defined as 16 GB divide by the maximum number of applications on one node, in this case, X will be 8 GB

Table 5.1: Workloads composed of multiple applications

Workloads	Hadoop Application	Spark Application
WL1	Pi, Sort, Grep, Word count	
WL2	Pi, Sort, Grep, Wordcount, TeraSort, TeraGen, PageRank, enhanced DFSIO	
WL3		Rpi, Sort, Wordcount, TeraSort
WL4		Rpi, Sort, Wordcount, TeraSort, Rpi, Sort, Wordcount, TeraSort
WL5	Sort, Wordcount	Sort, Wordcount
WL6	Pi, Sort, Grep, Wordcount	Rpi, Sort, Wordcount, TeraSort

Table 5.2: Input size of Hadoop Applications used

	Size(GB)
Pi	
Sort	63 GB
Grep	64.7 GB
Wordcount	64.7 GB
TeraSort	63 GB
TeraGen	
PageRank	
Enhanced DFSIO	read/write 30 GB

(our method) . For static partition, 16 GB for node manager. For Fair Sharing, it depends on the number of applications (if four applications, 4 GB for node manager, if eight applications, 2 GB each application). For Hadoop configuration parameters and Spark set up, the default values are used.

5.2 Experiment result

While Figures 5.1 and 5.2 show the result of only Hadoop applications with three different methods, Figures 5.3 and 5.4 depict three-way allocation of only Spark application. Figures 5.5 and 5.6 illustrate the running time of 3 approaches when running a combination of Hadoop and Spark applications. As seen from these figures, running time on Pi, Rpi, Teragen in three different methods were not significantly different. Data locality has little impact on these applications as they are computing intensive. In contrast, the differences for Sort, Grep, Wordcount,

Table 5.3: Input size of Spark Applications used

	Size(GB)
Rpi	
Sort	63 GB
Wordcount	64.7 GB
TeraSort	63 GB

Terasort, Pagerank and Enhanced dfsio were obvious. For these intensives, which were shown in Table 5.4, data locality profoundly affects Sort, Grep, Wordcount, Terasort, PageRank and enhanced dfsio. Data locality is truly the main factor for performance improvement.

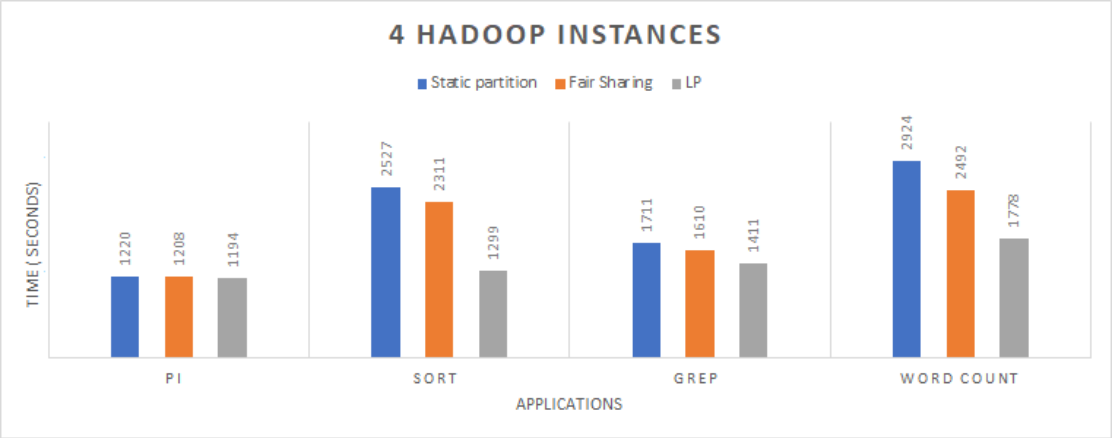

Figure 5.1: 4 Hadoop applications

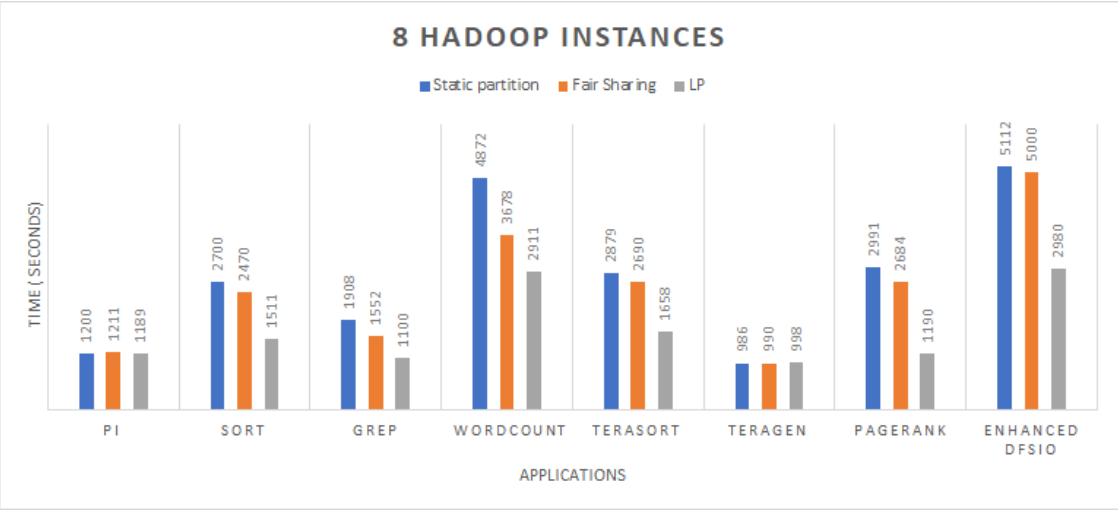

Figure 5.2: 8 Hadoop applications

Using LP and MCMF, a workload composed of four Hadoop applications has better performance

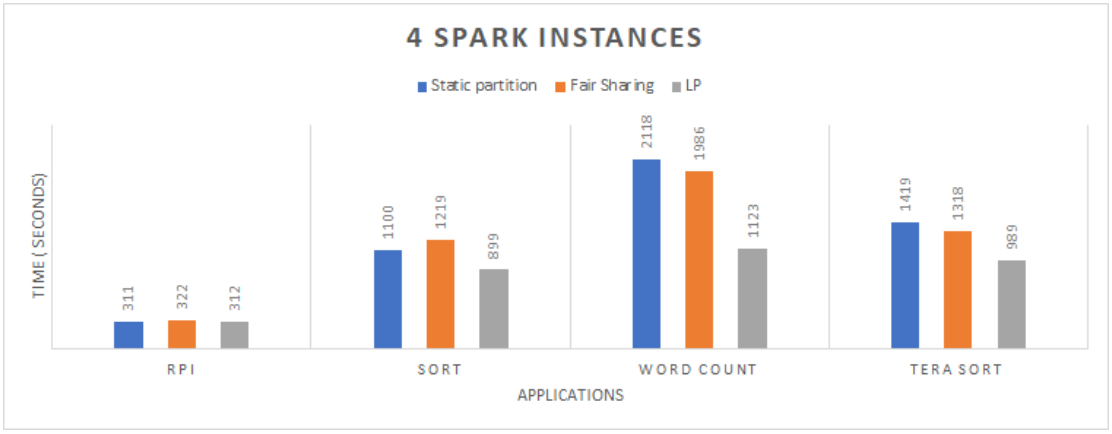

Figure 5.3: 4 Spark applications

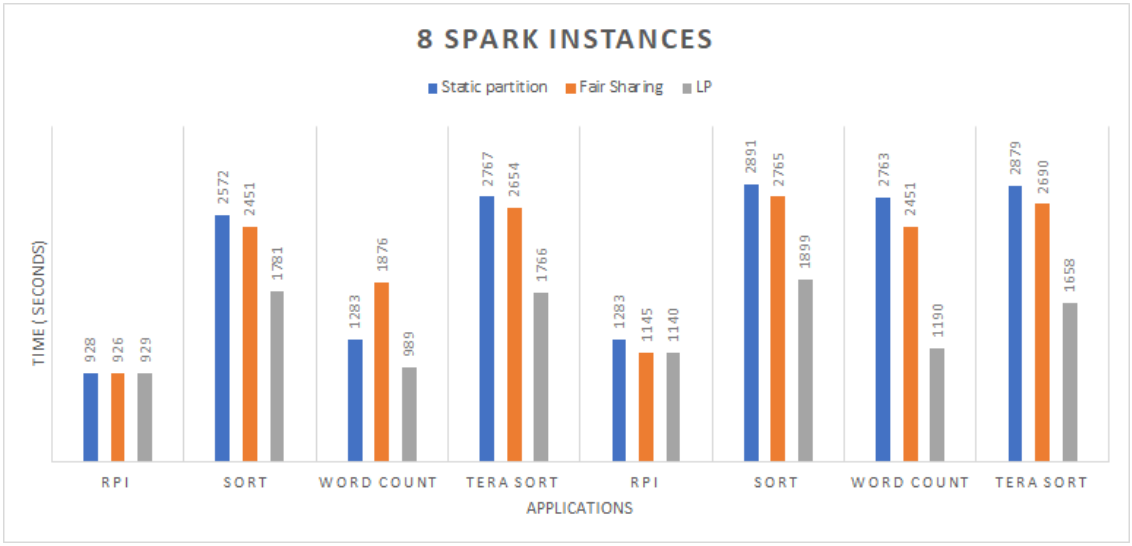

Figure 5.4: 8 Spark applications

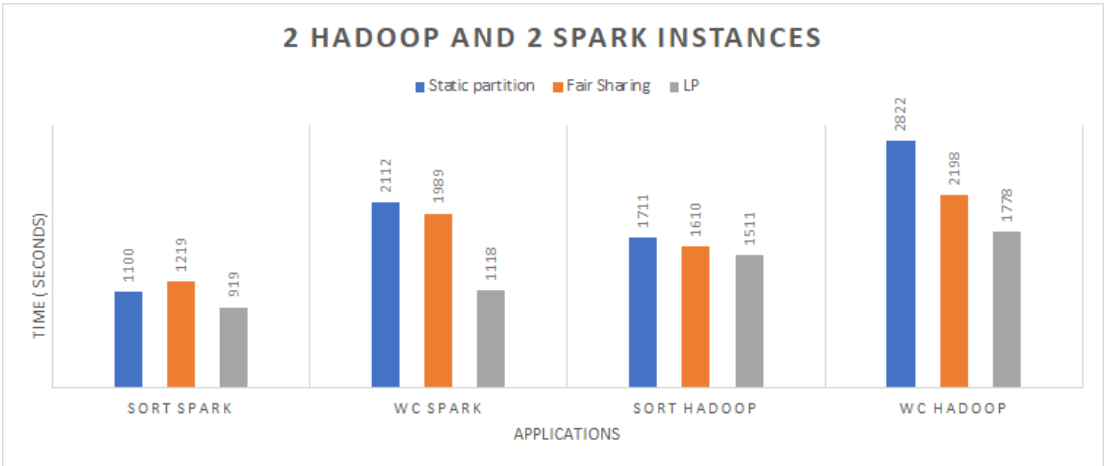

Figure 5.5: The combination of 2 Hadoop and 2 Spark applications

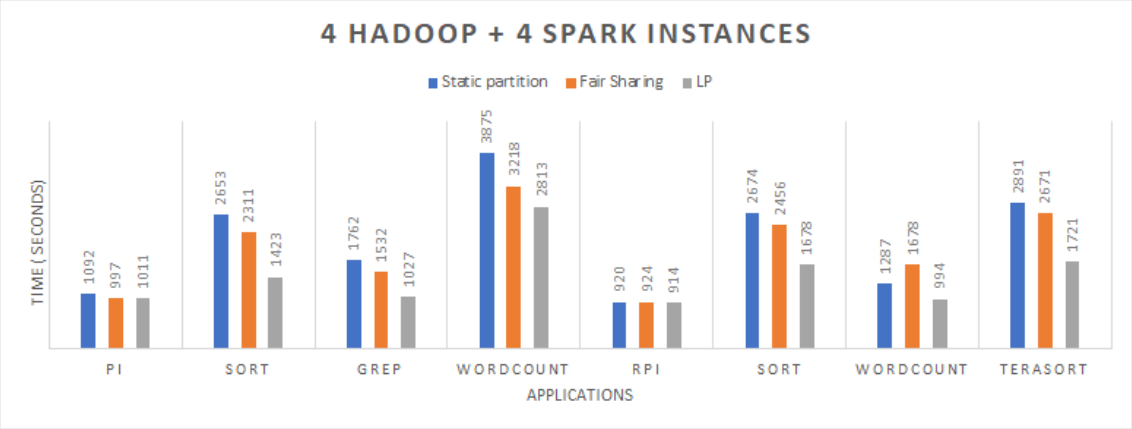

Figure 5.6: The combination of 4 Hadoop and 4 Spark applications

Table 5.4: The Characteristic of Applications

	Intensive
PiRpi	CPU
Sort	IO read & write
Grep	IO read
Wordcount	CPU & IO read
TeraSort	IO read & write
TeraGen	IO write
PageRank	CPU & IO read and write
Enhanced DFSIO	IO read and write

compared to fair sharing (1/4) and static partition with 37 % and 47 %, respectively. Similarly, the performance of a workload composed of eight Hadoop applications including Pi, Sort, Grep, Wordcount, TeraSort, TeraGen, PageRank and enhanced DFSIO improved 49 % and 67 % compared to fair sharing(1/8) and static partition, respectively. LP and MCMF also have better performance in a workload composed of four Spark applications compared to fair sharing (1/4) and static partition with 45 % and 48 %. The performance of a workload composed of eight Spark applications including RPi, Sort, Wordcount, and TeraSort with LP and MCMF improved 45.80 % and 48.9 % compared to fair sharing(1/4) and static partition, respectively. The performance of a workload composed of two Spark applications and two Hadoop applications including Sort and Wordcount which used LP and MCMF improved 31.73 % and 45.41 % compared to fair sharing(1/4) and static partition, respectively. It is the same in a workload composed of four Hadoop and four Spark applications, which improved 36 % and 48% faster than fair sharing (1/8) and static partition, respectively.

Both LP and MCMF have their own advantages. On one side, LP is faster on the condition that no new applications will arrive. MCMF, however, requires longer running time to map the input into graph format and then extract the solution. On the other side, MCMF takes an advance in dynamic scheduling because its graph state was already obtained and can be reused when the new application arrives.

Chapter 6

Conclusion

6.1 Summary of Thesis Achievements

In this work, we enhanced data locality on resource - shared clusters by implementing LP and MCMF to find the allocation solution. Our experiment result shows that our methods improve the performance of fair sharing and static partition, up to 49 % and 67 % in the case of eight Hadoop applications. Our methods can be applied in a really large cluster where input applications are imbalanced. However, our methods were just implemented on one replica factor and were assumed that the input file is independent of the applications.

6.2 Future Work

In the future, we will develop our methods for a higher replica, which will benefit in fault tolerance and also integrate our LP and Min Cost MCMF to Messos, a big resource - shared cluster framework.

Bibliography

[1] https://www.microsoft.com/en-us/research/project/dryad/?from=http

[2] http://hadoop.apache.org/.

[3] https://spark.apache.org/.

[4] https://mesos.apache.org/.

[5] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 127–144.

[6] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 261–276.

[7] B. Korte and J. Vygen, *Linear Programming Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 71–97.

[8] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[9] J. B. Orlin, "A polynomial time primal network simplex algorithm for minimum cost flows," *Mathematical Programming*, vol. 78, no. 2, pp. 109–129, Aug 1997.

[10] https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[11] J. C. Nash, "The (dantzig) simplex method for linear programming," *Computing in Science Engineering*, vol. 2, no. 1, pp. 29–31, Jan 2000.

[12] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer, 2006.

[13] F. A. Potra and S. J. Wright, "Interior-point methods," *Journal of Computational and Applied Mathematics*, vol. 124, no. 1, pp. 281 – 302, 2000, numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377042700004337

[14] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[15] M. Klein, "A primal method for minimal cost flows, with applications to the assignment and transportation problems," 1967.

[16] P. Tseng, I. of Technology. Laboratory for Information, M. Decision Systems, and D. Bertsekas, "Relaxation methods for minimum cost network flow problems," 01 1983.

[17] D. P. Bertsekas and P. Tseng, "Relaxation methods for minimum cost ordinary and generalized network flow problems," *Operations Research*, vol. 36, no. 1, pp. 93–114, 1988.

[18] A. V. Goldberg, "An efficient implementation of a scaling minimum-cost flow algorithm," *Journal of Algorithms*, vol. 22, no. 1, pp. 1 – 29, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S019667748570805X

[19] https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, professor Young-ri Choi of System Software Lab at UNIST, for giving me an opportunity to carry out my research project as well as a first step to becoming a researcher. She always gives me a myriad of useful advice. Without her support, I would not be able to discover my interesting research.

I would also like to express my thanks to Professor Beomseok Nam of Data Intensive Computing Lab at UNIST for his provision of a stable environment cluster so that I could accomplish this project. I would like to express a deep sense of gratitude to all professors in the Department of Computer Science at UNIST, from whom I have acquired valuable knowledge during the past two years.

I am also very thankful to all of my lab mates and Mr. Wonbae Kim, who not only gave me many comments and suggestions for my experiment but also assisted me during my pleasant stay in Korea.

Last but not least, I want to thank Mr. Mai Nguyen Trong Nhan for his volunteer assistance in proofreading of this thesis.

Finally, from the bottom of my heart, I would like to express my profound gratitude to my parents, especially to my father, who stands by my side with his endless love, encouragement and moral support.